Session 209

Audio Processing and Sequencing Services

Craig Keithley
USB & FireWire Technology Manager
Worldwide Developer Relations

Session 209

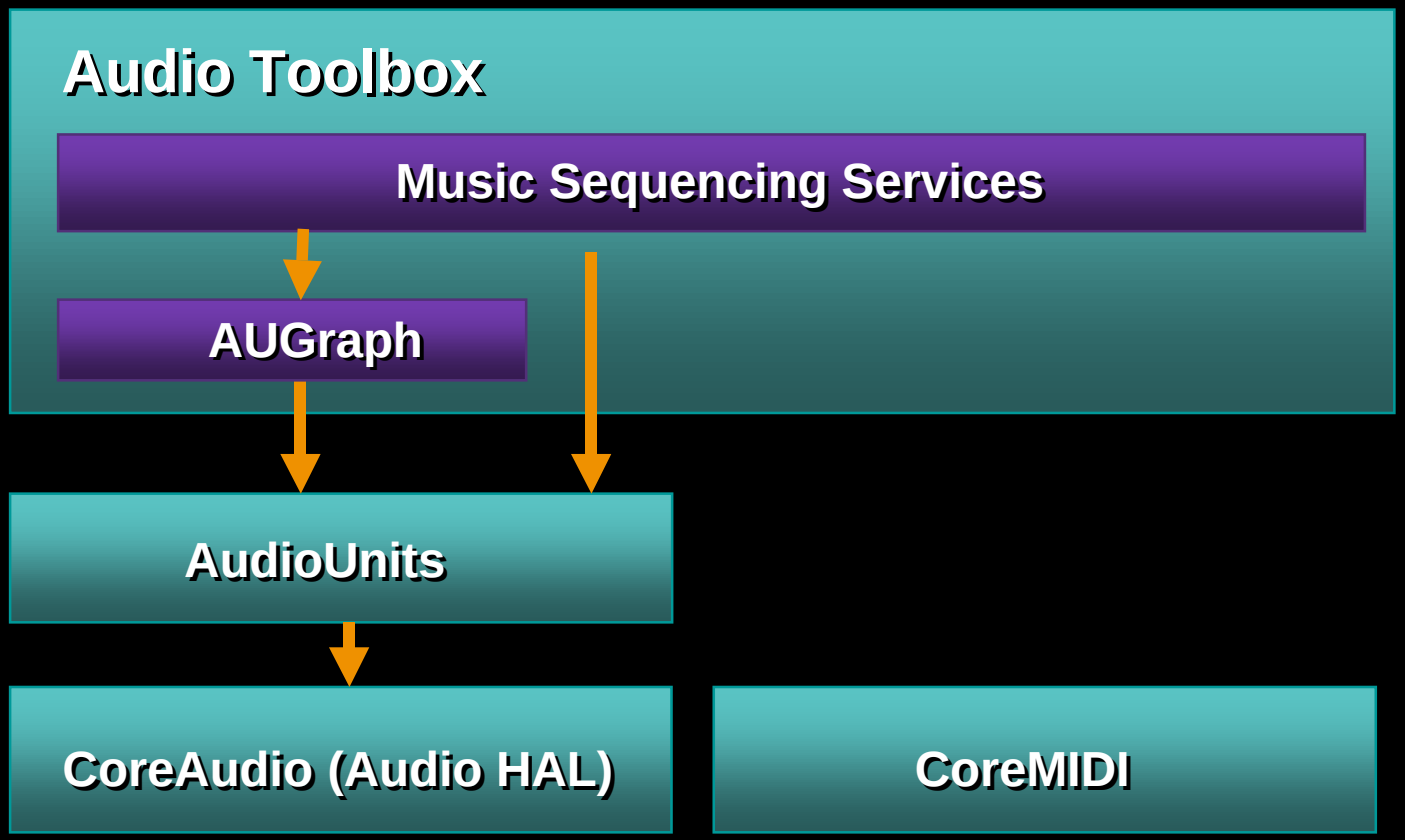
Audio Processing and Sequencing Services

Chris Rogers
Core Audio Engineer

CoreAudio Services

- AudioUnits
  - Components for processing audio
- AudioToolbox
  - Maintaining graphs of AudioUnits with the AUGraph APIs
  - Scheduling with Music Sequencing Services

Architectural Overview



AudioUnits Are Components

- Identified based on type/subtype/ID fields
  - Component Manager can query the set of available AudioUnits with FindNextComponent()
  - Instances
    - Created via OpenAComponent()
    - Released via CloseComponent().
- Refer to the Component Manager documentation for additional details

AudioUnit Component

- Generates, receives, processes, translates, or otherwise manipulates streams of audio
  - Useful as high-level building blocks
  - Can be used:
    - Singly
    - Connected together to form an audio signal graph (AUGraph)
- Apple provides a suite of AudioUnits
- Developers can implement their own

Kinds of AudioUnits

- Audio Sources
  - Software synthesizers (MusicDevice)
- Audio I/O
  - Hardware I/O devices
  - Direct from disk recording
  - Buffered audio I/O
- DSP Processors
  - Reverbs, Filters, Mixers, etc.

Kinds of AudioUnits

- Format converters
  - 16-bit integer to floating point
  - Interleavers/Deinterleavers
  - Sample Rate Converters
- Adapters
  - Input/output buffer size matchers
  - Deferred thread scheduler

Demo

AudioUnits

AudioUnit: Signal Flow

- N audio inputs and M audio outputs
- Audio DSP effects have both inputs and outputs

AudioUnit: Signal Flow

- An Audio Source such as a MusicDevice or input device might have only audio outputs

AudioUnit: Signal Flow

- An Audio Destination such as an output unit might have only audio inputs

AudioUnit Properties

- Very general and extensible mechanism for passing information to and from AudioUnits
  - Information is communicated via
    - Void* data parameter
    - Data byte-size parameter
- Type of information is identified by an AudioUnitPropertyID
- Similar to the MIDI and AudioHAL property APIs

Property Addressing

- Information is addressed to a particular section of an AudioUnit through scope and element
  - AudioUnitScope
    - Global
    - Input
    - Output
    - Group
  - AudioUnitElement is a zero-based index of a particular input, output, or group

Property Definitions

- AudioUnitPropertyIDs are defined in **<AudioUnit/AudioUnitProperties.h>** along with their associated data formats
- Additional properties will be added as needed
- Developers of AudioUnits
  - Free to define their own properties
    - Reserved range of AudioUnitPropertyIDs
  - Should support appropriate “system” properties

Example Property: Fast Dispatch

- The **kAudioUnitProperty_FastDispatch** property
  - Bypasses normal Component Manager dispatching
  - Allows the client to directly call a function pointer corresponding to time-critical AudioUnit functions
    - AudioUnitRenderSlice
    - Get/Set Parameter
  - Typically used when AudioUnits are connected together
- May be used as an optimization, but is not required

DLSMusicDevice and Sound Banks

- Setting the Sound Bank used by the DLSMusicDevice
  - SoundFont 2
  - DownLoadable Sounds 2 (DLS)
- myErr =

```
AudioUnitSetProperty (
    myMusicDeviceComponent,
    kMusicDeviceProperty_SoundBankFSSpec,
    kAudioUnitScope_Global,
    0,  //elementID
    &mySoundBankFileSpec,
    sizeof(FSSpec));
```

Parameters

- AudioUnitGet/SetParameter()
  - Allows real-time control of processing algorithms
  - Parameters are keyed by AudioUnitParameterID
  - Value is 32-bit floating-point
- AudioUnitSetParameter() includes a time-stamp
  - Allows an AudioUnit to schedule parameter changes within a rendered slice of audio

Parameter Information

- Published list of supported parameters via kAudioUnitProperty_ParameterList
- Individual parameter description is provided by kAudioUnitProperty_ParameterInfo
  - Parameter name
  - Units of measurement:
    - Rate, hertz, cents, decibels, linear gain, seconds, etc.
  - Min, max, and default value
  - Flags

Basic AudioUnit States

- AudioUnit states are
  - Open (Closed)
    - **OpenAComponent() / CloseComponent()**
  - Initialized (Uninitialized)
    - **AudioUnitInitialize() /AudioUnitUninitialize()**
  - Running (not Running!)
    - **AudioUnitStart()/AudioUnitStop()**

Opening and Initialization

- Opening
  - No significant resource allocations are expected to occur when the AudioUnit component is first opened (with OpenAComponent())
- Initialization
  - AudioUnitInitialize() is called after optional configuration has occured
  - This is where the AudioUnit allocates its memory, etc.

Running

- AudioOutputUnit
  - Represents the destination for all audio in a graph
  - After initialization,
    - **AudioUnitStart()/AudioUnitStop()**
  - Typically starts the processing of audio through several connected AudioUnits
- DefaultAudioOutput
  - Starts the hardware AudioDevice it is attached to

AudioUnitReset()

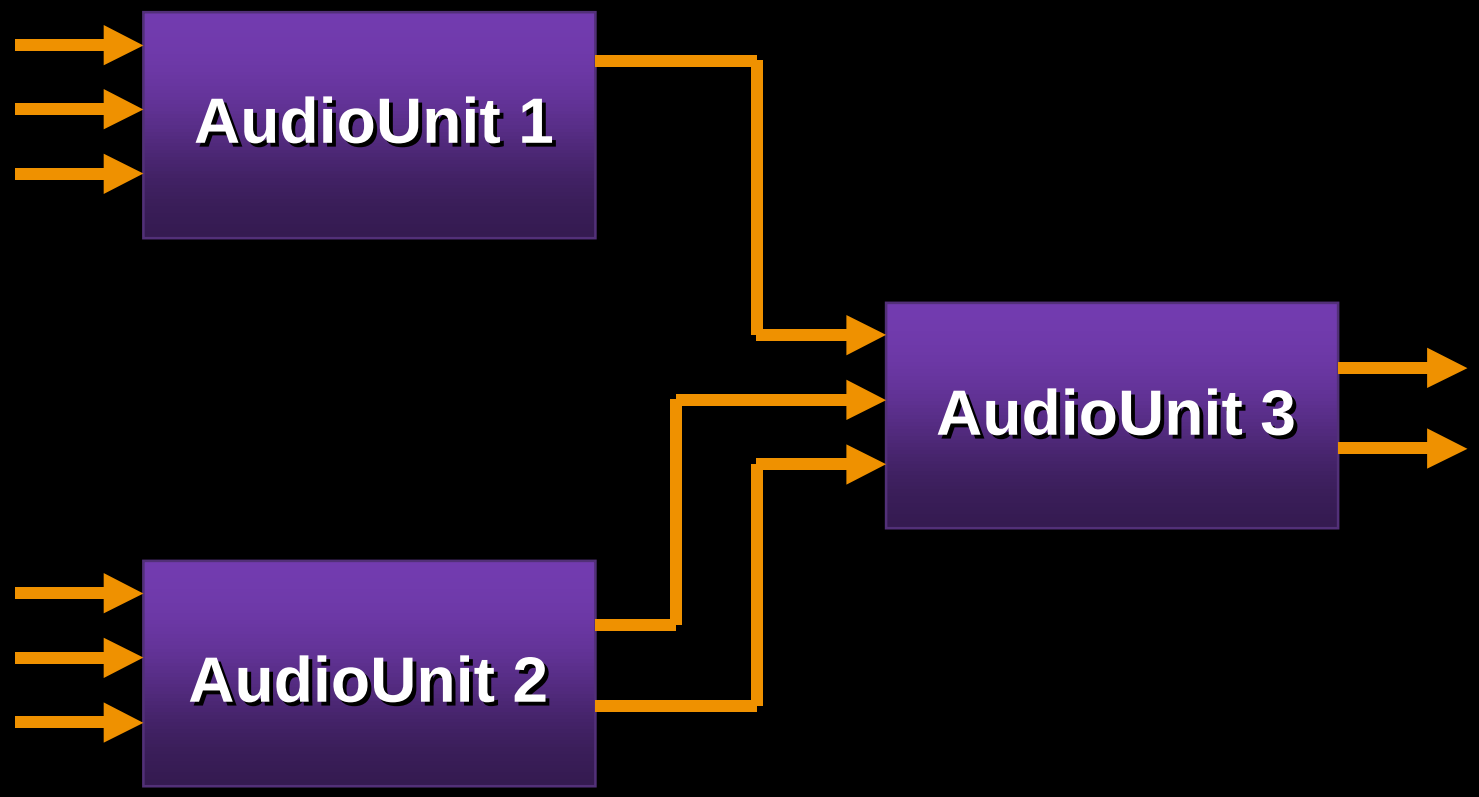
- AudioUnitReset() may be called on any initialized AudioUnit
- Resets the AudioUnit to its initialized state
  - Clears any buffers (delay or reverb memory)
  - Resets filter memory
  - Stops any playing notes (in a MusicDevice software synthesizer)

AudioUnit I/O Management

- Pull I/O model
- Specifies through its properties the number and format of its inputs and outputs
- Each input/output is a whole stream of N-interleaved audio (or side-band) channels (typically linear PCM floating point)

AudioUnit: Audio Connections

- May connect graphs of AudioUnits together



Input Connection Methods

- 1) Connect an AudioUnit output to an input using kAudioUnitProperty MakeConnection
  - Audio data is automatically routed to the input with no required user intervention
- 2) Register a client callback using kAudioUnitProperty_SetInputCallback
  - Client can provide audio source data to an AudioUnit through a user-defined callback

Flexibility in Making Connections

- Inputs may be connected according to either of these two methods on an independent basis
- AUGraph API provides a higher-level connection service, freeing the client from calling the AudioUnit directly

Output Audio Stream

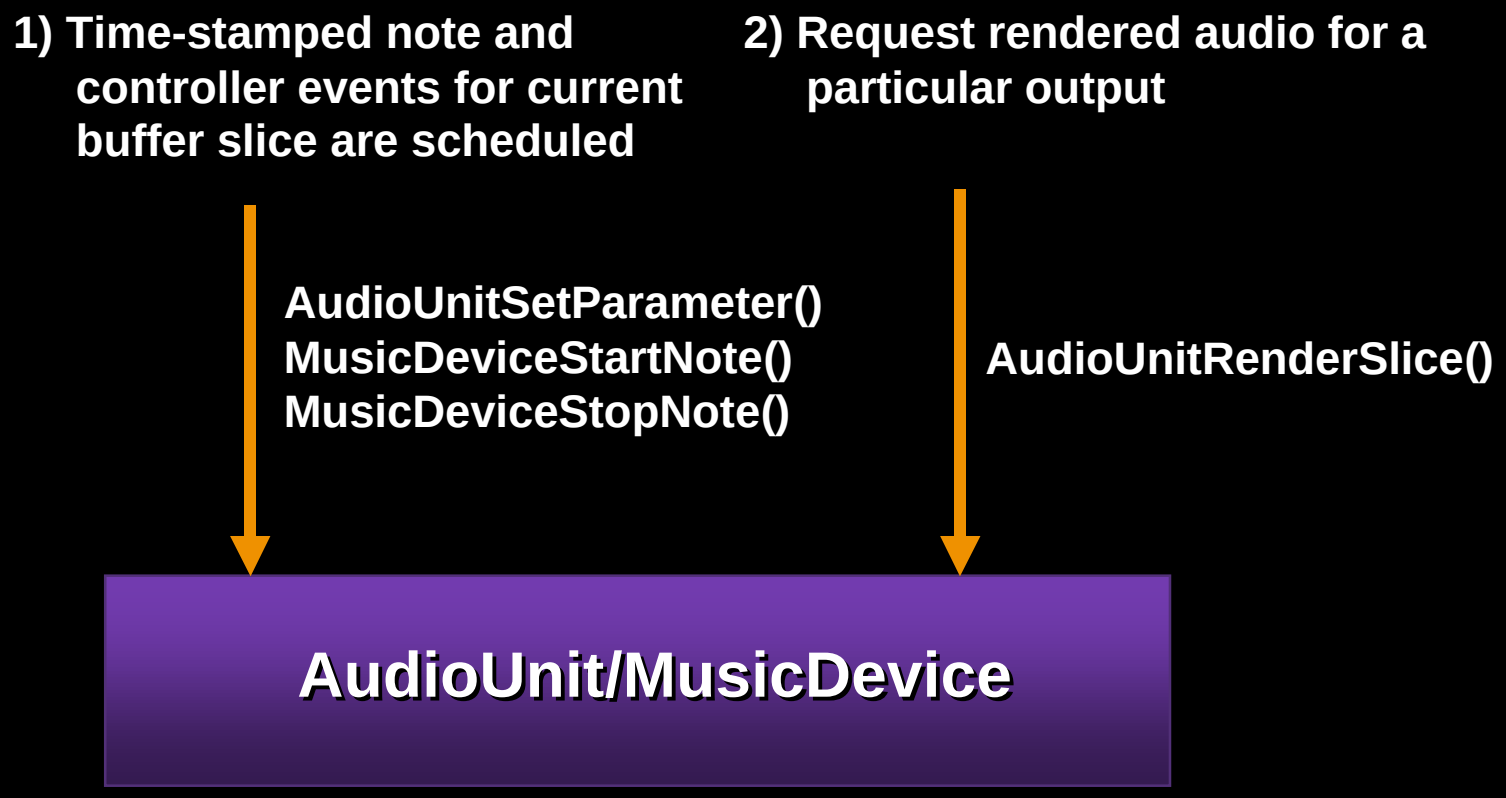
- AudioUnitRenderSlice()
  - "pulls" audio from a particular AudioUnit output
- AudioUnitRenderSlice() will often be called by another AudioUnit (the one whose input requires data)

AudioUnitRenderSlice() Arguments

- AudioTimeStamp
  - Specifies the start time of the buffer to be rendered,
    - Synchronizing host time of the machine with sample time of the audio to lock it with other realtime events such as MIDI
- AudioBuffer
  - Passes in and receives back a buffer of audio
  - Client may pass in a buffer or let the AudioUnit provide it

AudioUnit: Scheduling

- Two-phase Schedule/Render model



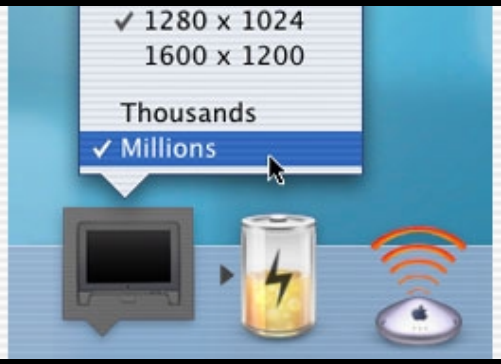
Pre/Post Rendering Notification

- AudioUnitSetRenderNotification() is a mechanism for a client to install a callback
- Allows client to perform arbitrary tasks every time AudioUnitRenderSlice() is called
  - Notification is provided both before and after the AudioUnit performs its rendering
- Example Usages:
  - Scheduling
  - CPU load measurement

How to Write an AudioUnit

- Written like any other component
- SDK soon to be available on Web site

AUGraph APIs



AUGraph APIs

- High-level representation of a set of AudioUnits along with the connections between them
- Construct arbitrary signal paths through which audio may be processed (modular routing system)
- Deals with large numbers of AudioUnits and their relationships

Why Use AUGraph APIs?

- Real-time routing changes allow for connections to be created and broken while audio is being processed
- Maintains representation even when AudioUnits are not instantiated

AUNode

- Represents an AudioUnit in an AUGraph
- The AudioUnit may not yet be instantiated if the AUGraph has not yet been opened, but the AUNode maintains enough information to create it

AUNode Construction/Destruction

- AUGraphNewNode()
  - Adds an AUNode to an AUGraph given a ComponentDescription
  - Optional “class-data” to initialize an AudioUnit with
- AUGraphRemoveNode()
  - Removes the AUNode from its AUGraph

AUGraph

- Connections
  - **AUGraphConnectNodeInput()**
  - **AUGraphDisonnectNodeInput()**
  - **AUGraphClearConnections()**
  - **AUGraphUpdate()for realtime changes**
- Inspection
  - **AUGraphGetNodeCount()**
  - **AUGraphGetIndNode()**
  - **AUGraphGetNodeInfo()**

AUGraph State

- Maintains its representation through AUNodes even when the AudioUnit components themselves are not instantiated
  - OpenComponent() has not yet been called
- States are
  - Open (Closed)
  - Initialized (Uninitialized)
  - Running
- Correspond directly with the AudioUnit states

AUGraph State APIs

- AUGraphOpen()/AUGraphClose()
  - AUGraphIsOpen()
- AUGraphInitialize()/AUGraphUninitialize()
  - AUGraphIsInitialized()
- AUGraphStart()/AUGraphStop()
  - AUGraphIsRunning()

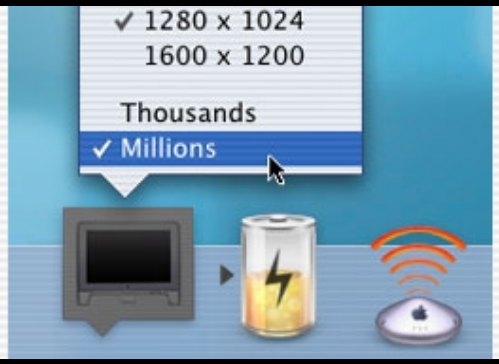
Persistence and “Class Data”

- **kAudioUnitProperty_ClassInfo** allows AudioUnits to persist their current configurations via “class data”
  - “Class data” may be supplied when an AUNode is added to the AUGraph
  - May be updated when the AudioUnit is closed
- Optional property

AUGraph "Head" AudioUnit

- Because AudioUnits follow a "pull model", there is usually one AudioUnit at the head of the graph
- "Head" AudioUnit is often an AudioOutputUnit and implements the **AudioUnitStart() / AudioUnitStop()** selectors
  - Called when AUGraph is started or stopped
    - AUGraphStart()/AUGraphStop()

Music Sequencing Services



Music Sequencing Services

- Simple API for construction/editing of multitrack MIDI sequences
- Events address
  - AudioUnits
  - MusicDevices
- Runtime for real-time playback of sequences

Music Sequencing Features

- Live edits (cut, copy, paste, merge, replace, etc.) of playing sequence
- Could be used as core sequencing engine for a sequencing application
- High precision scheduling uses units of “beats”
- Tempo map
- Reading/writing SMF
- Track attributes (mute, solo, looping, etc.)

MusicPlayer

- Provides the transport control:
  - **MusicPlayerSetTime()/MusicPlayerGetTime()**
  - **MusicPlayerPreroll()**
  - **MusicPlayerStart()/MusicPlayerStop()**
- Playback engine for a MusicSequence
- Synchronization between event-driven host-based audio processing and MIDI devices

MusicSequence

- Contains multiple playback tracks of event data (including MIDI)
- A track is called a MusicTrack
- An event is called a MusicEvent

MusicTrack

- Consists of an ordered sequence of time-stamped MusicEvents, plus properties:
- Destination (AUNode or MIDIEndpoint)
- Mute/solo
- Loop length, loop repetitions
- Time offset

MusicEvent Types

- kMusicEventType_MIDINoteMessage
  - MIDI note-on with duration (for note-off)
- kMusicEventType_MIDIChannelMessage
  - MIDI channel messages (other than note-on/off)
- kMusicEventType_MIDIRawData
  - System exclusive data

MusicEvent Types

- kMusicEventType ExtendedTempo
  - Tempo change in BPM
- kMusicEventType ExtendedNote
  - Like MIDI note-on—for advanced/esoteric apps
- kMusicEventType ExtendedControl
  - Calls AudioUnitSetParameter()

MusicEvent Types

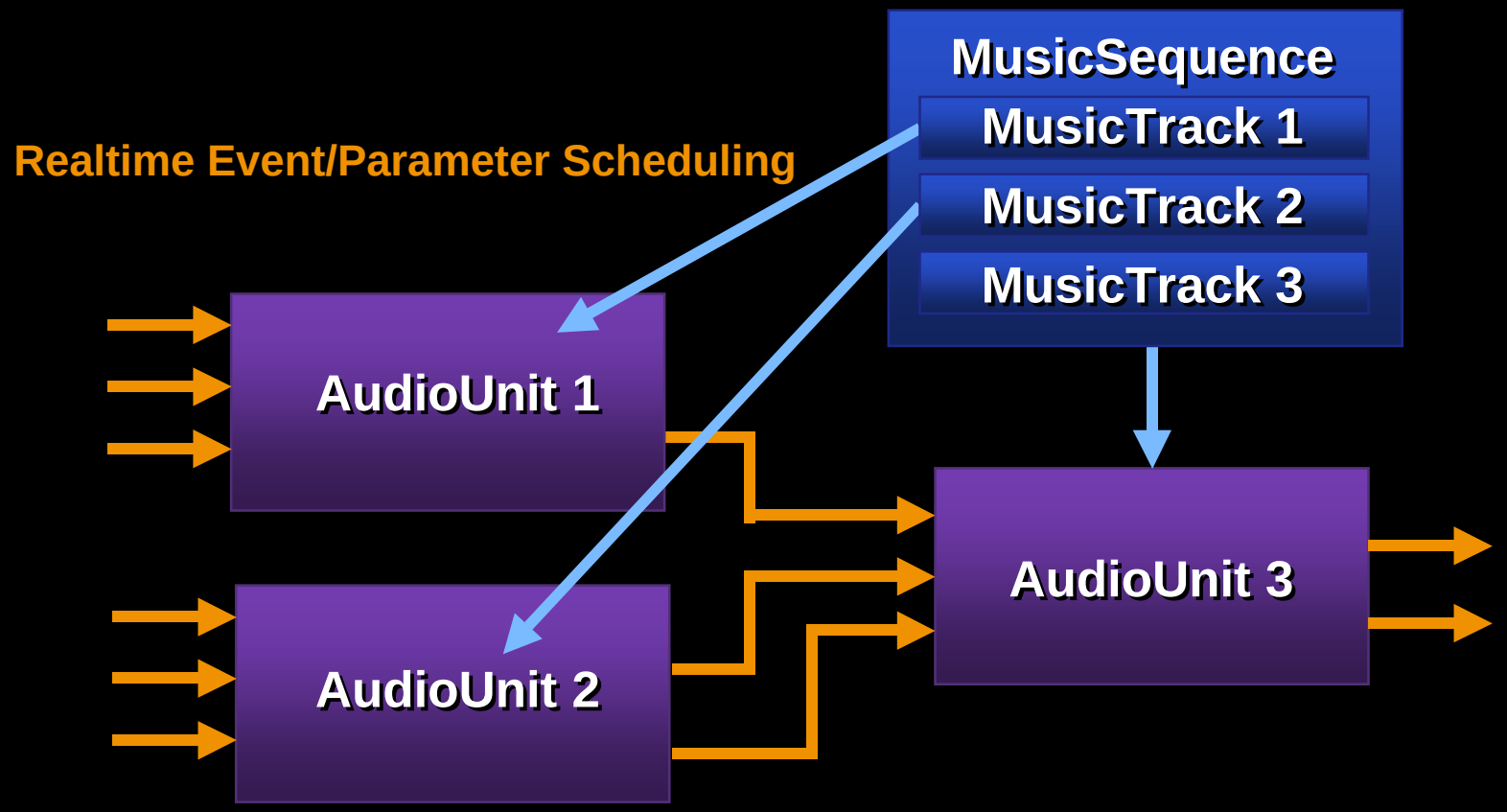
- kMusicEventType_User
  - User defined data
- kMusicEventType_Meta
  - Standard MIDI file meta event

Adding MusicEvents to MusicTracks

- There exist corresponding APIs for adding each of the MusicEvent types, for example:
  - **MusicTrackNewMIDINoteEvent(**
    MusicTrack inTrack,
    MusicTimeStamp inTimeStamp,
    const MIDINoteMessage *inMessage);

Sequencing With an AUGraph

- Sequence tracks address individual AudioUnits



Editing Operations on MusicTracks

- MusicTrackMoveEvents()
- NewMusicTrackFrom()
- MusicTrackClear()
- MusicTrackCut()
- MusicTrackCopyInsert()
- MusicTrackMerge()

MusicEventIterator

- Inspect and manipulate individual events within a MusicTrack
- Useful for generating piano roll, or event list type user-interface based on the contents of MusicTracks
- Allows developers to create custom editing operations by walking through all of the events in a given time range and manipulating them in arbitrary ways

MusicEventIterator

- Construction/Destruction
  - **New/DisposeMusicEventIterator()**
- Navigating a MusicTrack
  - **MusicEventIteratorSeek()**
  - **MusicEventIteratorNextEvent()**
  - **MusicEventIteratorPreviousEvent()**
  - **MusicEventIteratorHasPreviousEvent()**
  - **MusicEventIteratorHasNextEvent()**

MusicEvent Inspection

- MusicEventIteratorGetEventInfo()
  - Returns event type
  - Timestamp
  - Cast the returned void* pointer to its corresponding event data structure

MusicEvent Manipulation

- MusicEventIteratorDeleteEvent()
- MusicEventIteratorSetEventTime()

Demo

Using Music Sequence

Demo

MIDIGraph Demo

Core Audio Java

- Java Doc available for API
- Should understand the fundamental designs of core audio frameworks
  - Regardless of language used
- Java API presents the same functionality

Resources

Core Audio Mailing List

Specifications and SDKs

http://lists.apple.com

Developer Web Site

http://developer.apple.com/audio

Related Sessions

208 Audio Services in Mac OS X: Using Core Audio in Mac OS X	Room J2 **Thurs., 5:00pm**
136 Sound & Networking for Games: Enabling Games with Audio & Networking	Room J2 **Fri., 2:00pm**
210 MIDI in Mac OS X: MIDI Services Available in Mac OS X	Room J2 **Fri., 5:00pm**

Who to Contact

Apple Worldwide Developer Relations

Craig Keithley
FireWire & USB Technology Manager
keithley@apple.com

Developer Seeding

audio@apple.com

http://developer.apple.com/events/urls.html

Q&A

Craig Keithley
FireWire & USB Technology Manager
keithley@apple.com

http://developer.apple.com/events/urls.html

WWDC
Worldwide Developers Conference 2001